Remarks

Reconsideration of the application is respectfully requested in view of the foregoing amendments and the following remarks. Claims 1, 3-7, 9, 10, 12, 14-15, 17-21, 24, and 26-37 are pending in the application. No claims have been allowed. Claims 13, 22, and 23 have been canceled without prejudice. Claims 1, 12, and 14 are amended. Claims 1, 6, 12, 14, and 24 are independent. Claims 33-37 have been added.

Cited Art

The Office action dated Feb. 9, 2009 ("Action") applies the following cited art: U.S. Patent No. 6,560,774 to Gordon ("Gordon"); Lidin, Inside Microsoft .NET IL Assembler ("Lidin"); and U.S. Patent No. 6,412,020 to Leach et al. ("Leach").

Patentability of Claims 1, 3-7, 9-10, 12-24, and 26-32 Under 35 U.S.C. § 103(a) Over Gordon, Lidin, *and* Leach

The Action rejects claims 1, 3-7, 9-10, 12-24, and 26-32 under 35 U.S.C. § 103(a) as being unpatentable over *Gordon* in view of *Lidin* and further in view of *Leach*. The Action's rejection is respectfully traversed, for at least the following reasons:

Independent Claim 1 Is Allowable.

The Action rejects claim 1 for allegedly being unpatentable under 35 U.S.C. 103(a) over *Gordon* in view of *Lidin* in view of *Leach*. (Action, pg. 3). The Applicants disagree, but in the interest of expediting prosecution have amended claim 1 to recite:

> *translating the code segment from the programming language to one or more representations of a typed intermediate language, wherein the one or more representations of the typed intermediate language are capable of representing programs written in a plurality of different source languages, and wherein the one or more representations comprise a first object having a known type, and wherein the translating comprises:*
>
> *determining that the first object will be type-checked as an unknown type, and*
>
> *based on the determining, designating the first object as having the unknown type*

Support for this amendment can be found in the original application as filed. For example, see the original application at page 9, lines 11 through 25.

***Leach* does not teach or suggest translating code, including determining to type-check an object of known type as an unknown type.** The Action contends that "*Leach* further discloses IUnknown as a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known." Applicants disagree.

What *Leach* describes is the .NET Framework interface type named "IUnknown," which is what *Leach* calls an "interface named IUnknown (and referred to as the unknown interface or the object management interface)." (*Leach*, col. 6, lines 65-66). *Leach*'s IUnknown is "a function member that indicates which interfaces are implemented for the object." (*Leach*, col. 6, lines 63-65). The IUnknown interface is not used for type-checking a translated code segment; rather, IUnknown is used for what *Leach* describes "aggregation," which is defined by *Leach* as "the process of combining the capabilities of several distinct objects by enclosing their respective interfaces within an enclosing object." (*Leach*, col. 25, lines 34-37). Aggregation includes static aggregation and dynamic aggregation, but, for both forms of aggregation, the IUnknown type is determined in advance by a programmer using a programming language. "Using static aggregation, a programmer decides, in advance, which of its aggregate object interfaces the enclosing object should expose and then implements the QueryInterface method of the controlling IUnknown of the enclosing object to return pointers to these exposed interfaces when requested." (*Leach*, col. 25, lines 43-48). "In a preferred embodiment, dynamic aggregation is implemented using a multitype object. A multitype object is an object capable of aggregating objects of varying types, hence its name. Only interfaces that have been coded such that they are capable of being aggregated can be enclosed within a multitype object." (*Leach*, col. 26, lines 6-12).

Therefore, the IUnknown interface described by *Leach* must be coded by a programmer to be of the IUnknown type. Hence, *Leach* does not teach or suggest translating a code segment from a programming language to an intermediate language representation, "wherein the one or more representations comprise a first object having a known type, and wherein the translating comprises determining that the first object will be type-checked as an unknown type, and based on the determining, designating the first object as having the unknown type," as recited by amended claim 1.

As stated in the Action, this deficiency is not cured by *Lidin* or *Gordon*. (Action, pg. 4). Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether considered alone or in any combination thereof, the method of amended claim 1, the Action has not sufficiently stated a case for obviousness. For at least these reasons, the allowance of amended claim 1 is respectfully requested.

Independent Claim 6 Is Allowable.

The Examiner rejects claim 6 over *Gordon* in view of *Lidin* further in view of *Leach*. (Action, pg. 5). The Examiner's rejection is respectfully traversed.

***Gordon* does not describe determining whether to retain type information for elements of an intermediate language representation.** The Action contends that *Gordon* describes for each intermediate language representation "determining whether to retain type information for one or more elements of the representation," and "based on the determination, associating one or more elements of the representation with a type," citing *Gordon*, FIG. 23; column 27, lines 4–33; column 1, lines 54-62; column 6 line 58 through column 7 line 16; column 26 lines 5-14. (Action, pg. 5). Applicants disagree.

What FIG. 23 and the cited text describe is what *Gordon* calls a "Virtual Execution System" (VES). (*Gordon*, col. 3, lines 7–8; col. 27, lines 12–13). *Gordon* recites that a VES can skip the step that *Gordon* calls "verification" for trusted code, and that the VES can also skip verification of precompiled native code that is fully trusted. (*Gordon*, col. 27, lines 12–33). "Verification" apparently includes checking IL code metadata and code for consistency and accuracy, including semantic checks such as "reference aspect checks (such as byref and refany checks), value class checks, native-size primitive type checks, and tail call verifications." (*Gordon*, col. 6, line 58 through col. 7, line 5). *Gordon* teaches that the VES makes assumption about code are made to aid verification: "the type of the arguments to any method in the code being verified *are fixed*. . . . This means that dynamic type information for arguments *do not need to be maintained* on a per-basic block basis." (*Gordon*, col. 7, lines 57–60) (emphasis added). *Gordon* further teaches that type information for primitive locals is "fixed" and that for non-primitives, type information must be maintained:

> *[T]he type of primitive local variables, such as integers, floating points, etc., is fixed.* ***If a variable is declared as being a primitive type, then it is not***

> ***allowed to store a non-primitive type**, such as an object reference, into it. In this manner, a single bit--"live" or "dead" for example--is sufficient to convey a primitive variable's state at any point. For non-primitive variables, however, complete type information must be maintained.*

(*Gordon*, col. 7, line 66 through col. 8, line 6) (emphasis added). Therefore, *Gordon* states that the variable type for what *Gordon* calls "primitive types" (*e.g.*, integers or floating points) is fixed and that a variable, which is of a primitive type, "is not allowed to store a non-primitive type, such as an object reference." (*Gordon*, col. 7, lines 66 through col. 7, line 6). Hence, *Gordon* states that the type of primitive local variables is fixed, and further states that for some arguments, dynamic type information is not maintained at all; *Gordon* drops all type information for a variable; *Gordon* does not describe converting the type information to a different type, which would be type-checked.

Therefore, *Gordon*'s VES does not teach or suggest the language of independent claim 6, "for each representation, determining whether to retain type information for one or more elements of the representation; [and] based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type."

***Leach* does not teach or suggest selectively retaining type information, including determining to type-check an object of known type as an unknown type.** As discussed above regarding amended claim 1, *Leach* describes the IUnknown interface pointer, which is hand-coded by programmers to implement object aggregation. Hence, *Leach* does not teach or suggest selectively retaining type information, including "a type, designated as an unknown type, indicating the element can be of any type" as recited by claim 6. As stated by the Action, this deficiency is not cured by *Gordon* or *Lidin*. (Action, pg. 6). Applicants will not further belabor the patentability of claim 6 regarding this point.

***Gordon* cannot be combined with *Leach* to result in the claimed arrangement.** The Action contends that *Leach* discusses "IUnknown as a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known." (Action, p. 6). Applicants disagree, but even assuming for the sake of argument that this is true (and, as Applicants discussed above regarding claim 1, it is not), *Gordon* and *Leach* cannot be combined to result in the arrangement recited by claim 6.

As discussed above, *Gordon* describes a verification system in which "primitive types" (e.g., integers or floating points) are fixed, and are "not allowed to store a non-primitive type, such as an object reference." (*Gordon*, col. 7, lines 66 through col. 7, line 6). Even assuming that *Leach* actually describes an unknown type equivalent to claim 6 (and it does not), *Leach* cannot be combined with *Gordon* to form a verification system that uses an unknown type. This is because *Gordon* states that the types of primitive variables is "fixed" and are "not allowed to store a non-primitive type." *Gordon* in view of *Leach* therefore does not teach or suggest, and in fact teaches away from, the method of claim 6, which recites: "determining whether to retain type information for one or more elements of the representation" and "based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type."

Gordon's deficiencies are not cured by *Lidin* or *Leach*. Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether considered alone or in any combination thereof, the method of claim 6, the Action has not sufficiently stated a case for obviousness. For at least these reasons, the allowance of claim 6 is respectfully requested.

Independent Claim 12 Is Allowable.

The Action rejects independent claim 12 as being allegedly unpatentable under 35 U.S.C. § 103 over *Gordon*, *Lidin*, and *Leach*. (Action, pg. 7). Applicants disagree, but in the interest of expediting prosecution have amended claim 12 to recite:

> *replacing the types associated with the plurality of programming languages with the types of the intermediate language, wherein the types of the intermediate language comprise plural programming language specific primitive types associated with the plurality of programming languages and a type designated as an unknown type, wherein the type designated as the unknown type has size information associated with it, wherein at least one of the plural programming language specific primitive types is replaced with the unknown type, wherein the size information comprises size information of a machine representation of the type designated as the unknown type.*

Support for this amendment can be found in the original application as filed; for example at page 14, lines 14-25; page 8, line 26 through page 9, line 25. The Action states that *Gordon* describes "replacing the types associated with the plurality of programming languages with the types of the intermediate language wherein the types of the intermediate language comprise

general categories of the types associated with the plurality of programming languages and a type designated as an unknown type." (Action, pg. 7) (citations omitted). However, *Gordon* also states that "the type of the arguments to any method in the code being verified are fixed. . . . the type of primitive local variables, such as integers, floating points, etc. is fixed. If a variable is declared as being a primitive type, then it is not allowed to store a non-primitive type." (*Gordon*, col. 7, line 57- col. 8, line 6). Hence, *Gordon* does not teach or suggest replacing primitive types with an unknown type; in fact, *Gordon* states that the type of primitive local variables is fixed.

Therefore, *Gordon* does not teach or suggest, and in fact teaches away from, the language of amended claim 12, which recites "replacing the types associated with the plurality of programming languages with the types of the intermediate language, wherein the types of the intermediate language comprise plural programming language specific primitive types associated with the plurality of programming languages" and "wherein at least of the plural programming language specific primitive types is replaced with the unknown type." *Gordon*'s deficiencies are not cured by *Lidin* or *Leach*.

***Leach* does not teach or suggest selectively retaining type information, including determining to type-check an object of known type as an unknown type.** As discussed above regarding claims 1 and 6, *Leach* describes the IUnknown interface pointer, which is hand-coded by programmers to implement object aggregation. Hence, *Leach* does not teach or suggest "wherein at least one of the plural programming language specific primitive types is replaced with the unknown type" as recited by claim 12. As stated by the Action, this deficiency is not cured by *Gordon* or *Lidin*. (Action, pg. 8). Applicants will not further belabor the patentability of claim 12 regarding this point.

Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether considered alone or in any combination thereof, the method of claim 12, the Action has not sufficiently stated a case for obviousness. For at least these reasons, the allowance of claim 12 is respectfully requested.

Independent Claim 14 is Allowable.

The Action rejects independent claim 14 as being allegedly unpatentable under 35 U.S.C. § 103 over *Gordon*, *Lidin*, and *Leach*. (Action, pg. 9). Applicants disagree, but in the interest of expediting prosecution have amended claim 14 to recite:

> *a type-checker module, wherein the type-checker is configured for applying the one or more rule sets to the elements of the intermediate representation, wherein the type-checker module selectively retains type information for some elements of the intermediate representation and selectively does not retain type information for at least one element of the intermediate representation by replacing a type associated with the at least one element with the type, designated as the unknown type, indicating the at least one element can be of any type*

The Action contends that "*Gordon* discloses the system selectively retains type information for some elements of the intermediate representation and selectively does not retain type information for other elements of the intermediate representation (e.g., col. 17:53 – col. 18: 28; col. 19: 60 – col.20: 10)." (Action, pg. 11). Applicants disagree.

The cited portions of *Gordon* describe what *Gordon* calls "Metadata Wellformedness," where "[f]or each method within a type . . . conditions such as the following are checked." (*Gordon*, col. 17, lines 52-54). *Gordon* then describes several conditions (*e.g.*, access rights, return type, unique name, etc.) that are checked. (*Gordon*, col. 17 line 55 through col. 18, line 28). *Gordon* also discusses "IL Type Verification" for "value class constructors." (*Gordon*, col. 19, line 60 through col. 20, line 10). *Gordon* therefore describes checking conditions for methods within a type, but does not mention replacing type information for any elements of an intermediate representation at all.

For at least these reasons, *Gordon* does not teach or suggest, and in fact teaches away from, the system of amended claim 14, which recites a type-checking module configured for checking one or more rule sets, including "not retain[ing] type information for at least one element of the intermediate representation by replacing a type associated with the at least one element with the type, designated as the unknown type." This deficiency is not cured by *Lidin* or *Leach*.

Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether considered alone or in any combination thereof, the system of amended claim 14, the Action has

not sufficiently stated a case for obviousness. For at least these reasons, the allowance of amended claim 14 is respectfully requested.

Independent claim 24 Is Allowable.

The Action rejects independent claim 24 as being allegedly unpatentable under 35 U.S.C. § 103 over *Gordon*, *Lidin*, and *Leach*. (Action, pg. 9). The Action's rejection is respectfully traversed, for at least the following reasons:

***Leach* does not teach or suggest representing types in an intermediate language including an unknown type.** *Leach* does not teach or suggest the method of claim 24, which includes "defining a plurality of types to be associated with elements of the intermediate language, wherein one of the plurality of types indicates that an element of the intermediate language is associated with a type designated as an unknown type." As discussed above regarding claims 1, 6, and 12, what *Leach* describes the IUnknown interface, which must be hand-coded in a programming language by a programmer. In fact, *Leach* does not teach or suggest using its IUnknown interface in an intermediate language at all.

Therefore, *Leach* does not teach or suggest, and in fact teaches away from, the method of representing types in an intermediate language claim 24, which recites "wherein one of the plurality of types indicates that an element of the intermediate language is associated with a type designated as an unknown type; wherein the type indicating that an element of the intermediate language is associated with the type designated as the unknown type has a size associated with it." Further, as stated by the Action, neither *Gordon* or *Lidin* describe "a type designated as an unknown type, where the unknown type indicates that an element of the representation is of a type that is not known." (Action, pg. 12).

Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether considered alone or in any combination thereof, the method of claim 24, the Action has not sufficiently stated a case for obviousness. For at least these reasons, the allowance of claim 24 is respectfully requested.

Dependent Claims 3-5, 7, 9-10, 15, 17-21, and 26-32 are Allowable.

Claims 3-5, 7, 9-10, 15, 17-21, and 26-32 are dependent from one of the independent claims discussed above. For at least the same reasons discussed above, and further based on

each claim's unique and non-obvious combination of features, dependent claims 3-5, 7, 9-10, 15, 17-21, and 26-32 are also allowable over the art of record.

Interview Request

If the claims are not found by the Examiner to be allowable, the Examiner is requested to call the undersigned attorney to set up an interview to discuss this application.

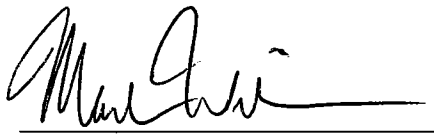
Conclusion

The claims in their present form should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By 

Mark W. Wilson
Registration No. 63,126